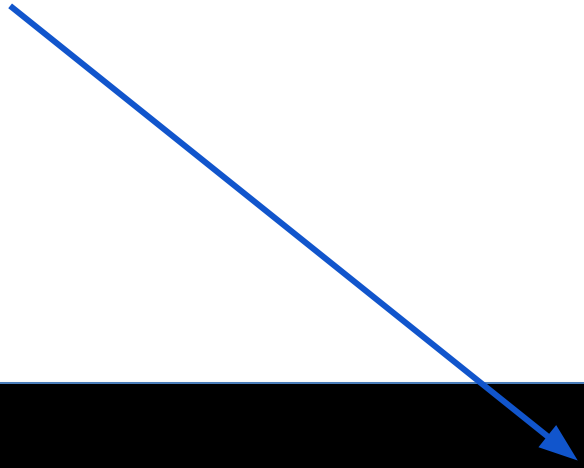
CSCI 3753: Operating Systems

Week 3
September 12, 2025

University of Colorado **Boulder**

Announcements

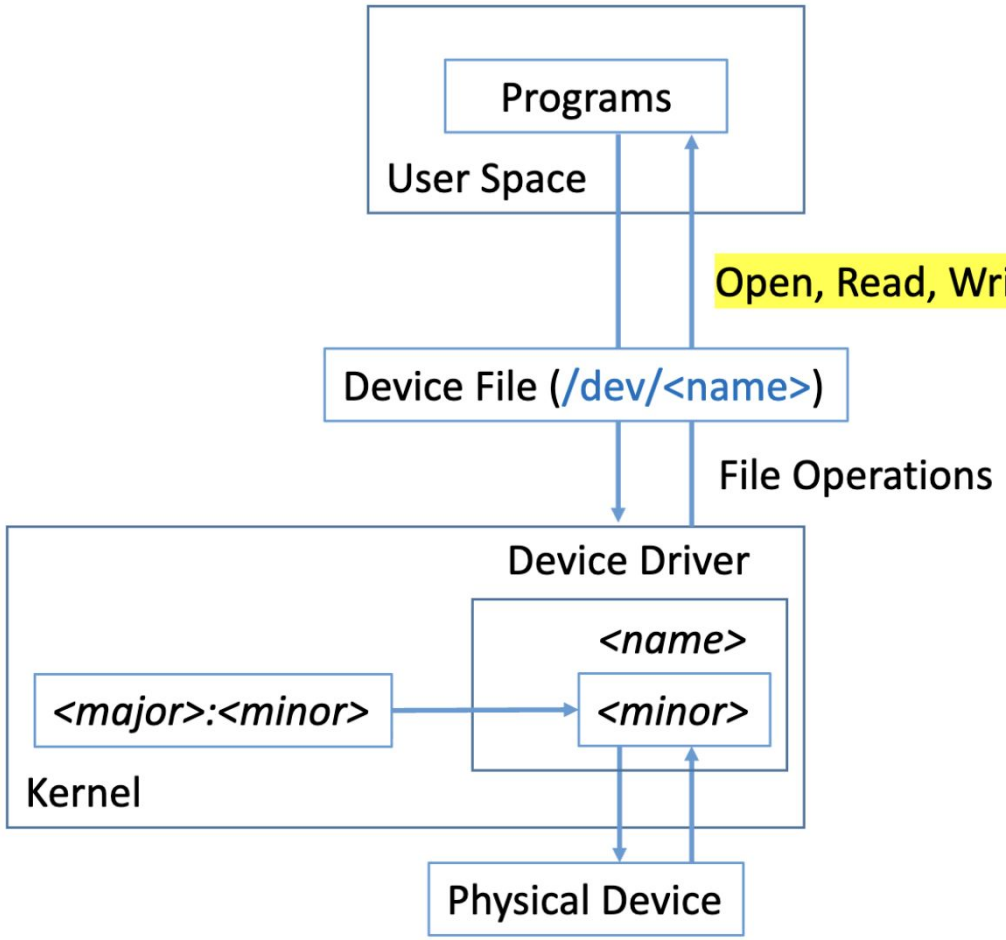
- PA0+1 Interview Grading almost over!
- Quiz 3 due at midnight
- Problem Set 1 due on Sunday
- PA2 and PA3 are SIGNIFICANTLY MORE WORK THAN PA0 and PA1
- Recitation materials: **https://tinyurl.com/CSCI3753**



Recap

- PA0 ✅
- PA1 ✅
- User/Kernel Space
- System Calls
- LKM

PA 3



mknod –m <permission>
<device_file_location>
<type of driver>
<major number>
<minor number>

PA 3

- create a Device Driver Module (LKM)
- implement file operations
  - open, seek, read, write, release
- make and load the module
- create a Device File for this Device

PA 3

- create a Device Driver Module (LKM)
- implement file operations
  - open, seek, read, write, release
- make and load the module
- create a Device File for this Device
- **create a test program**

PA 2

PA 2

- Test program for PA 3
- pa2test.c
  - infinite loop with the following features
    - r - read()
    - w - write()
    - s - seek()
      - SEEK_SET
      - SEEK_CUR
      - SEEK_END
    - control+d for termination
    - other entries should be ignored

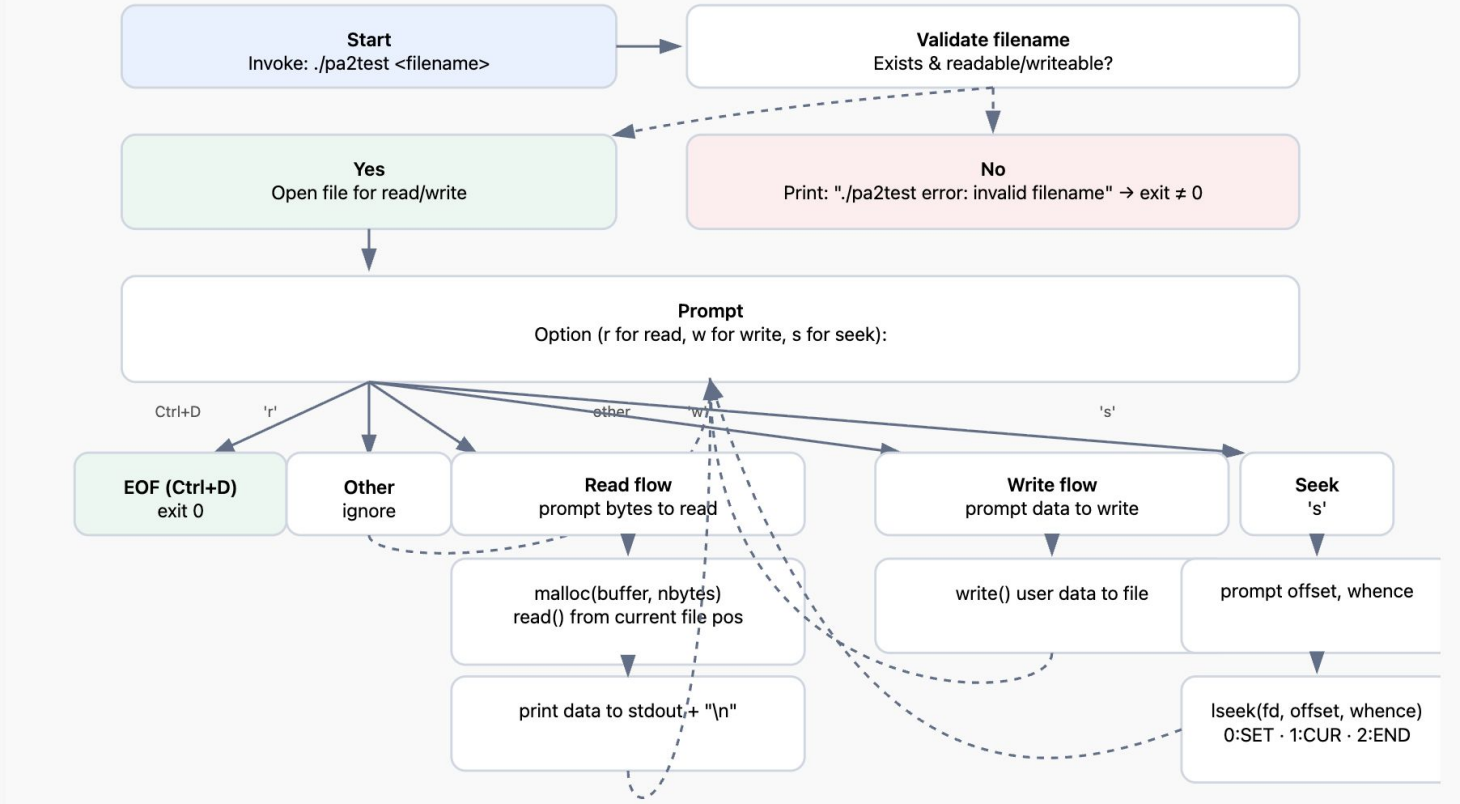
PA 2 Flow

Exit codes: `0` on normal (Ctrl+D) termination · `≠0` on error (e.g., invalid filename).

Seek *whence*: `0 → SEEK_SET`, `1 → SEEK_CUR`, `2 → SEEK_END`.

Loop behavior: unknown/other input is ignored; prompt repeats.

Reads use a `malloc()` 'd buffer sized to requested bytes.



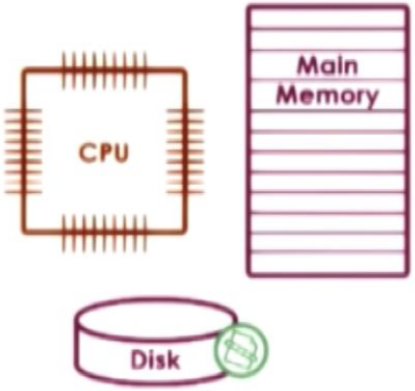
PA2 Demo

- Available on Canvas

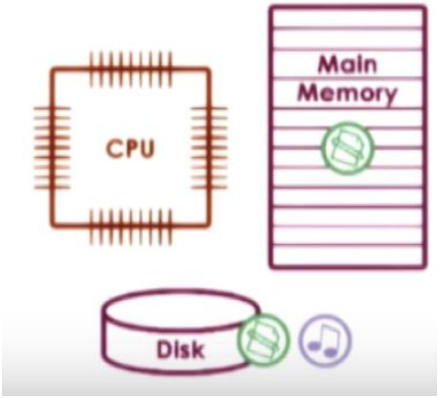
Processes and Threads

What is a Process?

OS manages hardware on behalf of applications

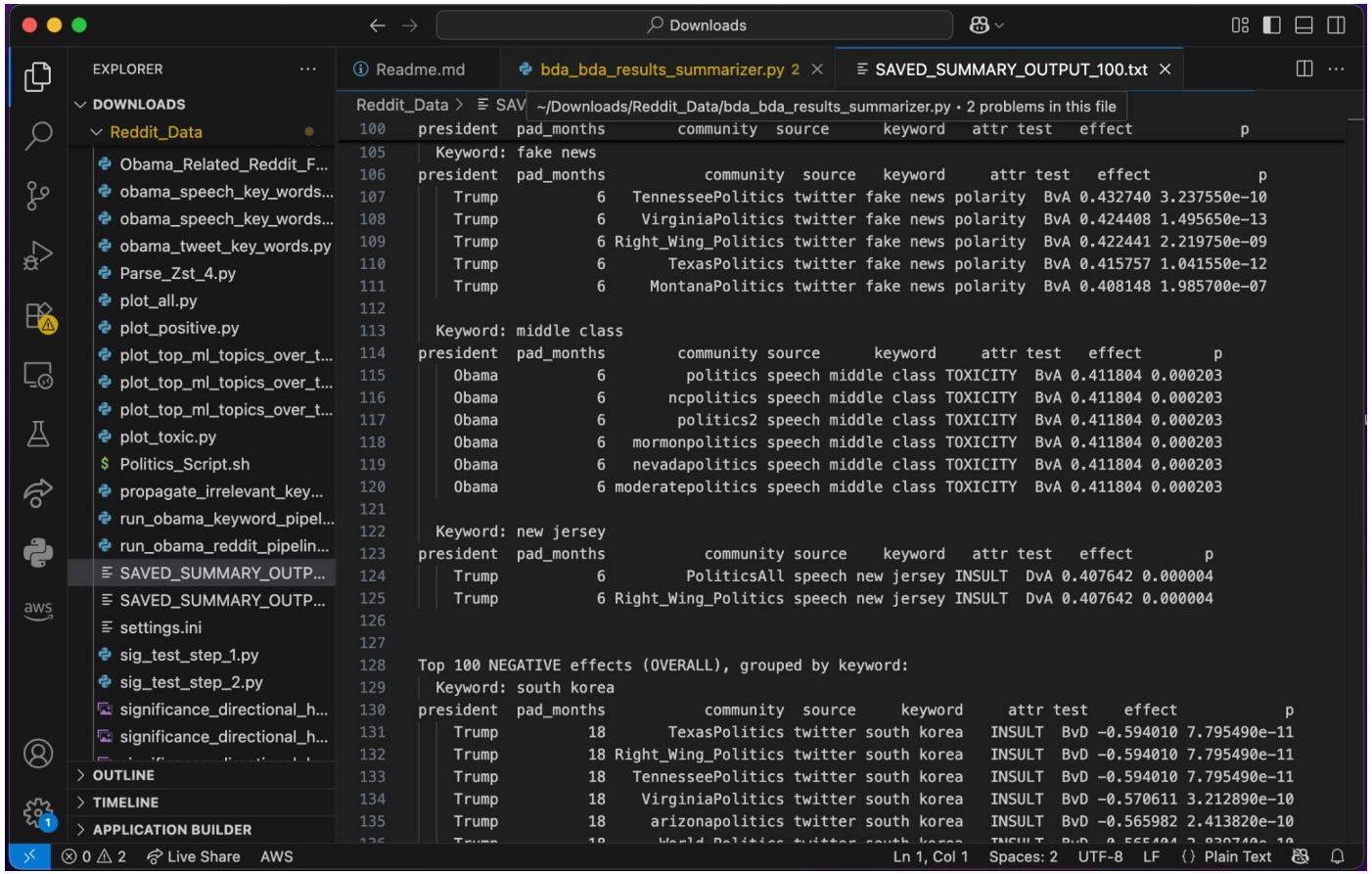


Program is an application on disk/flash memory (static)



Process is a program when loaded into main memory (active)

Example of a Process



How does the OS track Processes?

- Program Counter
- CPU Registers
- Stack Pointer

pc

```
sum = 0;
for (int i = 0; i < 10; ++i) {
  sum += i;
}
```

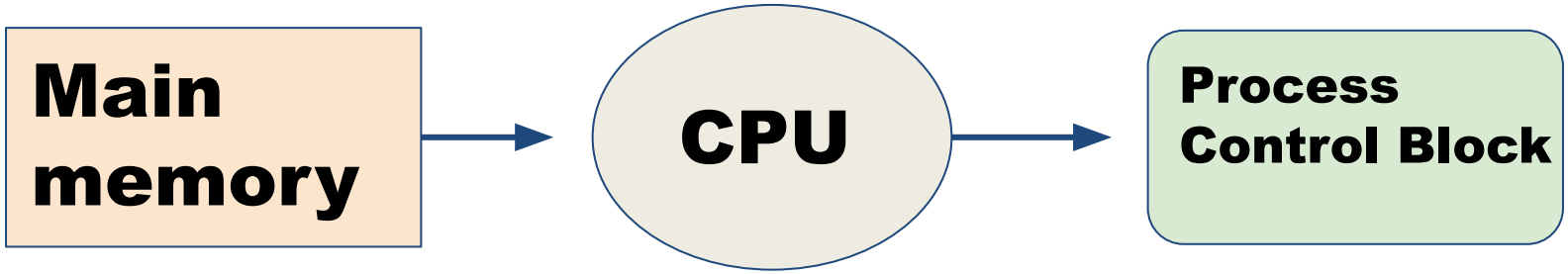
(a) Simple Loop Code

```
4004b8:     movl    $0x0,-0x8(%rbp)
4004bf:     movl    $0x0,-0x4(%rbp)
4004c6:     movl    $0x0,-0x4(%rbp)
4004cd:     jmp     4004d9 <main+0x25>
4004cf:     mov     -0x4(%rbp),%cax
4004d2:     add     %cax,-0x8(%rbp)
4004d5:     addl    $0x1,-0x4(%rbp)
4004d9:     cmpl    $0x9,-0x4(%rbp)
4004dd:     jle     4004cf <main+0x1b>
```

(b) Assembly Code

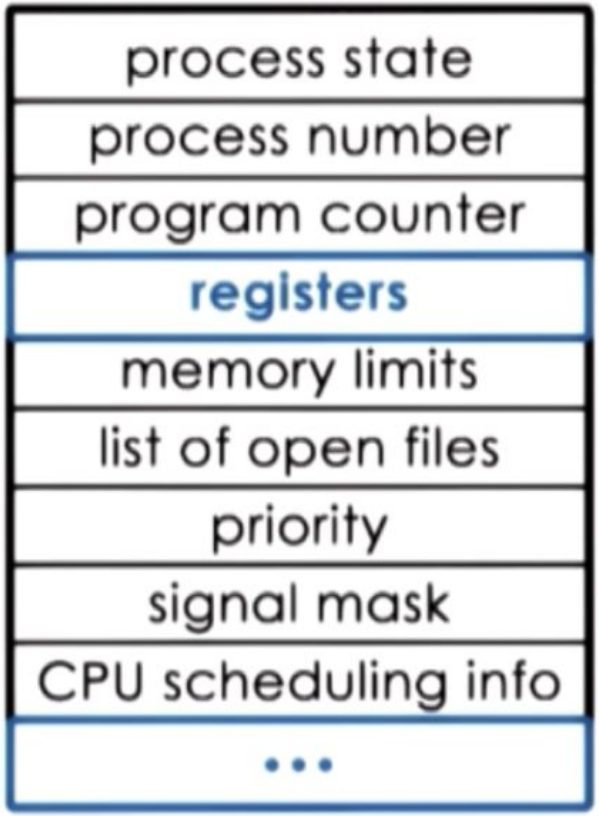
stack pointer

register

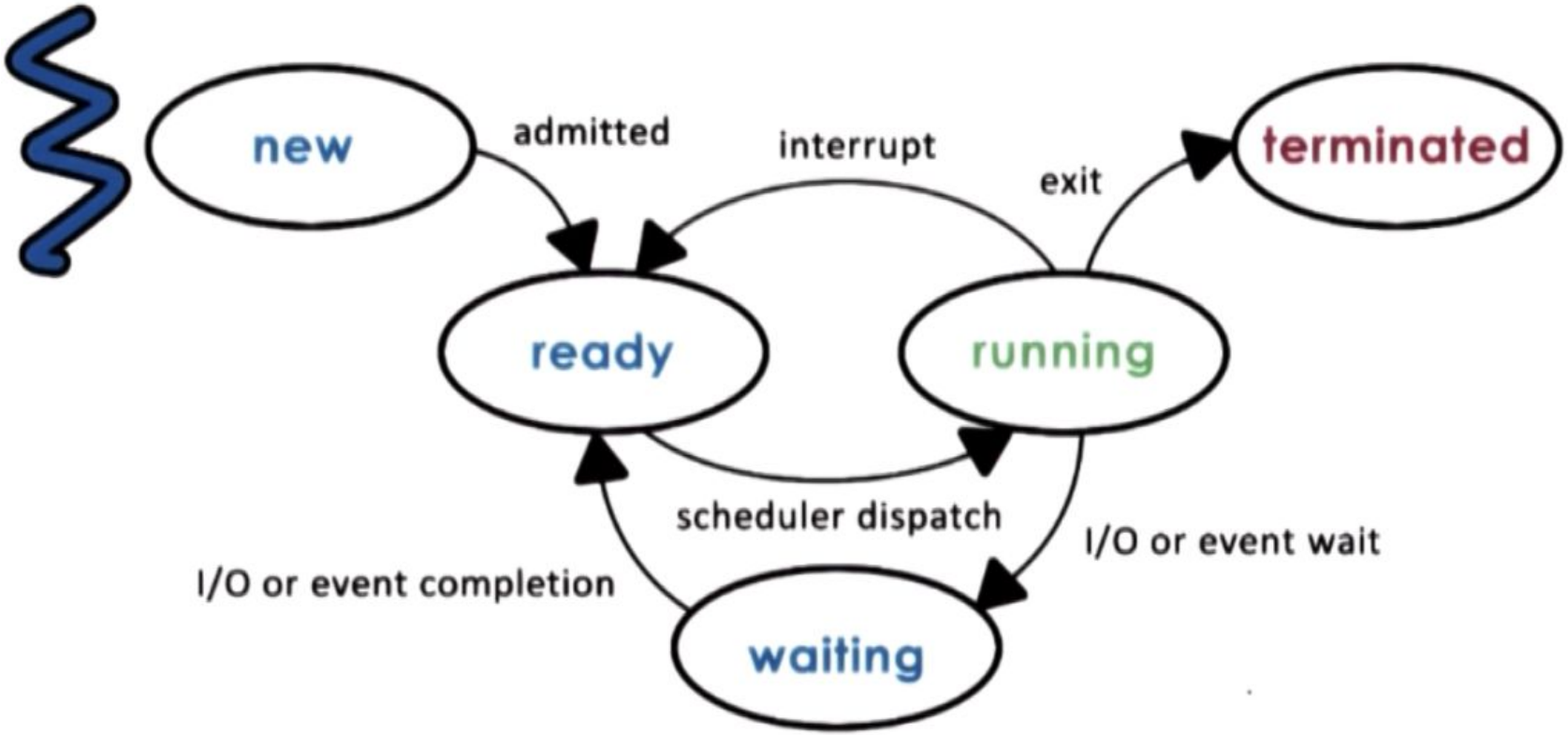


Process Control Block

- PCB created when process is created
- Certain fields are updated when process state changes
- Other fields change very frequently



Process Lifecycle: States



Process Lifecycle: States

- **New:** The process is being created
- **Running:** Instructions are being executed
- **Waiting:** The process is waiting for some event to occur (such as an I/O completion or reception of a signal)
- **Ready:** he process is waiting to be assigned to a processor
- **Terminated:** The process has finished execution

Processes

- Have an ID associated with it call the process ID (PID)

```
$ ps

PID        TTY      STAT   TIME          CMD
41230    pts/4    Ss       00:00:00     bash
51224    pts/4    R+       00:00:00     ps
```

- **ps** command
  - PID: process ID
  - TTY: controlling terminal associated with the process
  - STAT: process status code
  - TIME: total CPU usage
  - CMD: name of executable/command

/proc directory

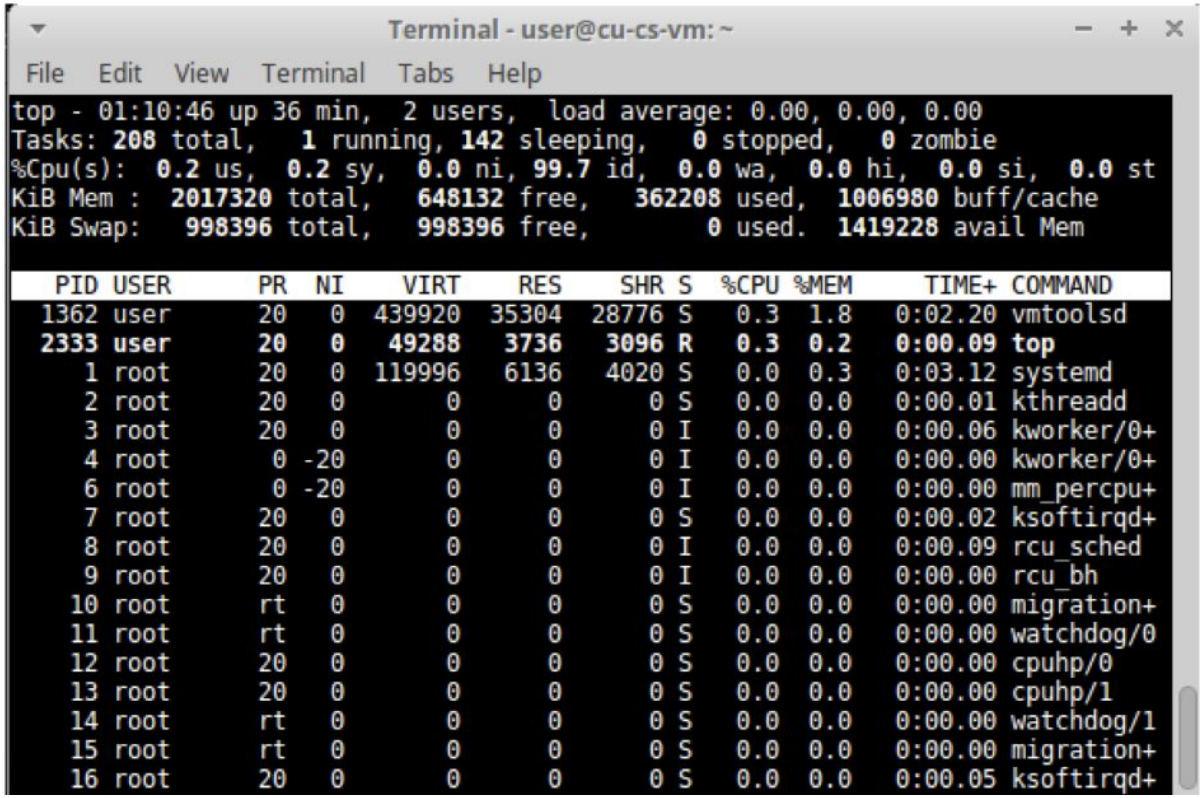
- Contains virtual files and numbered directories corresponding to each running process
- Directory name = process ids
- When a process ends, its directory in /proc disappears automatically
- File in a number directory:
  - cmdline
  - cwd
  - environ
  - fd
  - maps, statm, mem, stat, status

/proc directory

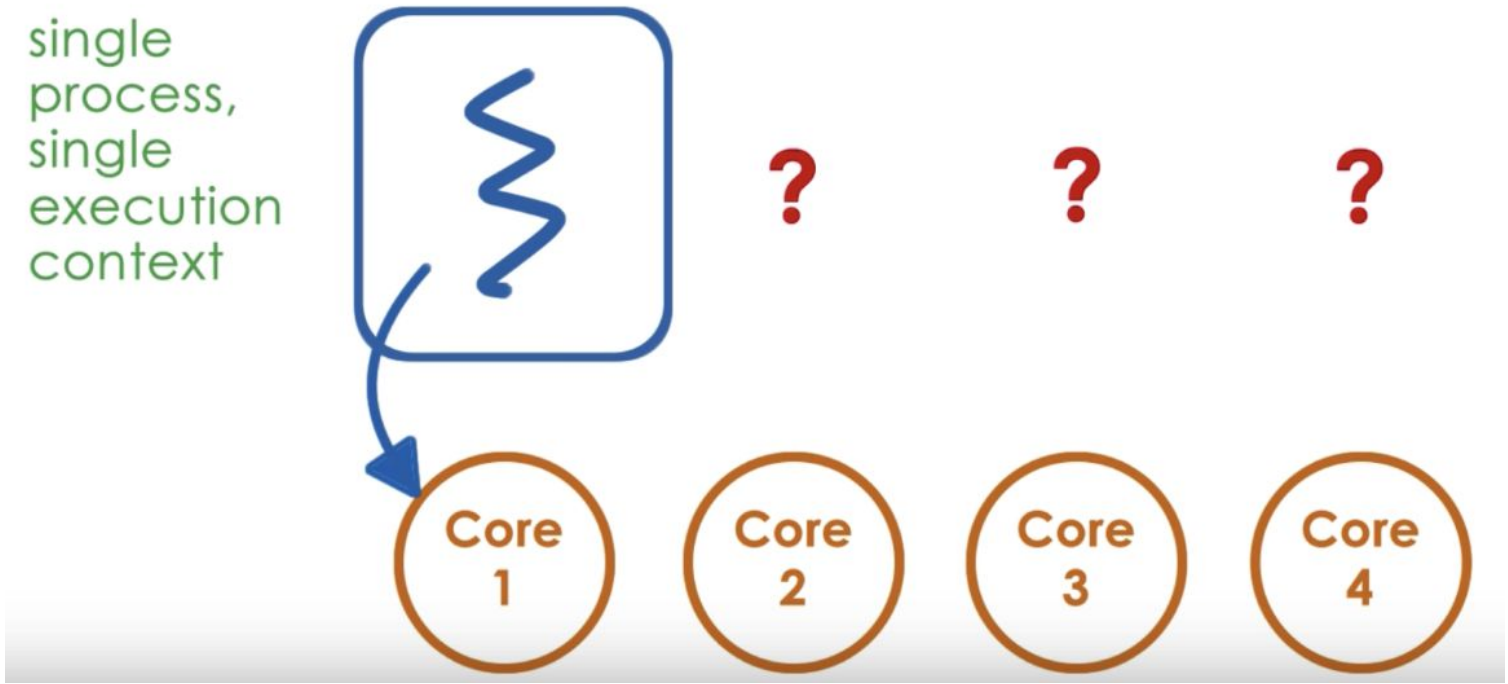
- Some typical virtual files that provide
  - Hardware information:
    - /proc/cpuinfo: identifies the type of processor used by your system
    - /proc/iomem: shows you the current map of the system's memory for each physical device
    - /proc/meminfo
    - /proc/interrupts
  - File info
    - /proc/filesystems:displays a list of the file system types currently supported by the kernel
    - /proc/partitions
  - Kernel info
    - /proc/cmdline: shows the parameters passed to the kernel at the time it is started
    - /proc/sys

top command

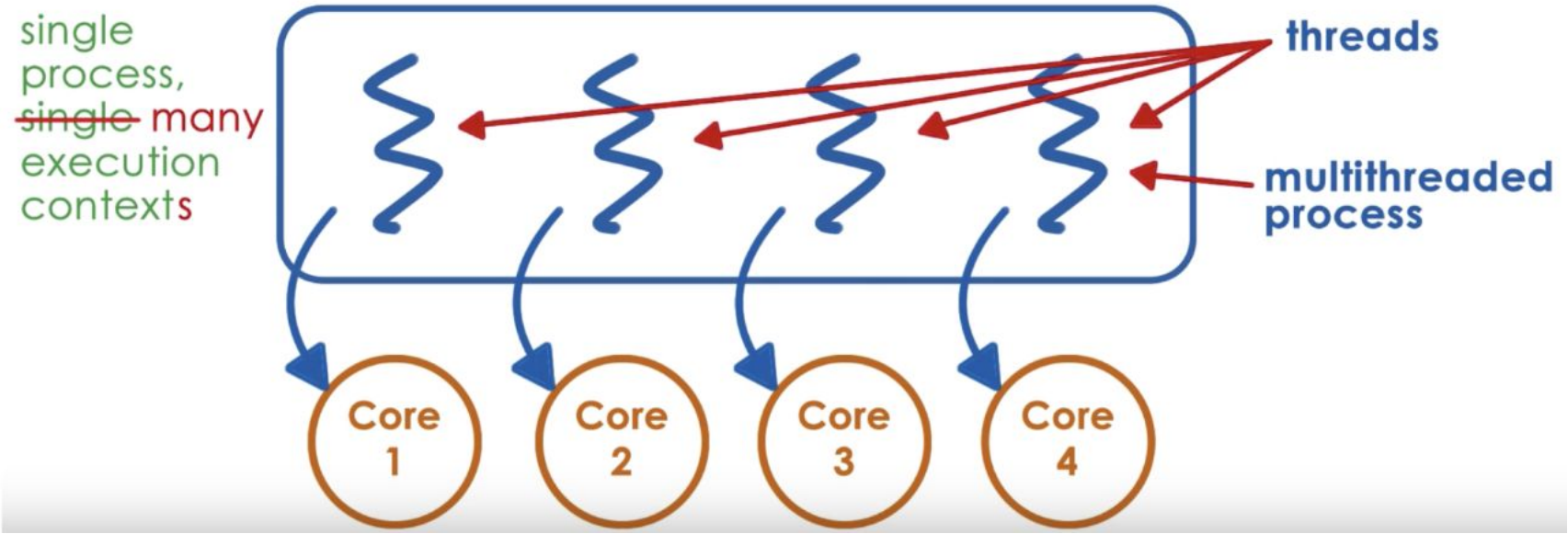
- Displays a list of processes with their resource usage



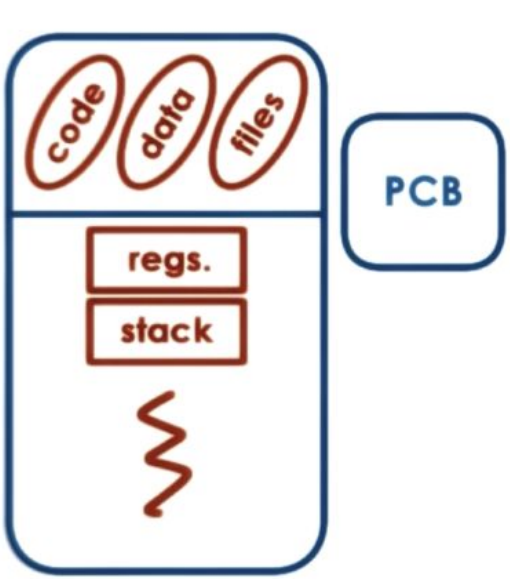
What if we have multiple CPUs?

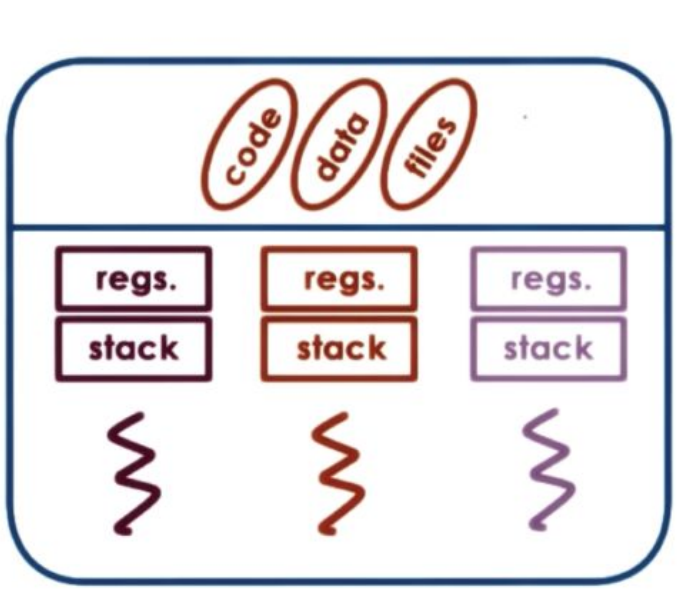


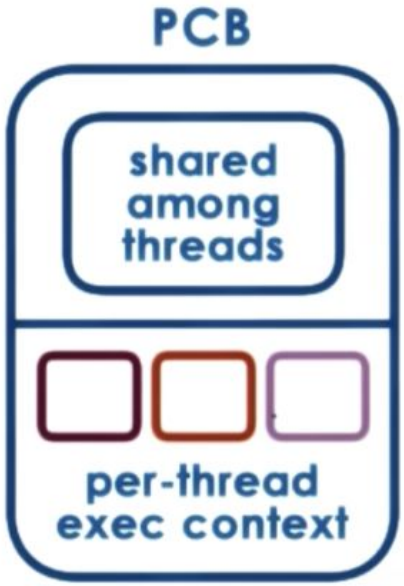
What if we have multiple CPUs?



Process vs Threads







Process vs Thread

Process	Thread
Executing instance of an application	Path of execution within a process
Used for heavy weight tasks	Used for small tasks
Takes more time for termination and creation and context switching	Takes less time for termination and creation and context switching
Process consume more resources.	Threads consume less resources
Process is isolated	Threads share memory

Threads

- Threads share the same address space:
  - Code
  - Data
  - Other OS resources
    - heap segments
    - open file descriptors
    - signal and signal handlers
    - current working directory
    - user and group id
- Threads each have their own unique:
  - Thread ID
  - Stack
  - Set of registers
  - Program counter

Common Pitfalls with Multithreading

- Race Conditions
- Thread Safe Code
- Mutex deadlock

Common Pitfalls with Multithreading

- Race Conditions
  - While code may appear on the screen in the order you wish the code to execute, threads are scheduled by the operating system and are executed at random
- Thread Safe Code
- Mutex deadlock

Common Pitfalls with Multithreading

- Race Conditions
- Thread Safe Code
  - There should be no static or global variables which other threads may clobber or read assuming single threaded operation.
  - Many non-reentrant functions return a pointer to static data → Thread unsafe functions
- Mutex deadlock

Common Pitfalls with Multithreading

- Race Conditions
- Thread Safe Code
- Mutex deadlock
  - Occur when a mutex is applied but then not “unlocked”
  - Cause program execution to halt indefinitely

Threads Demo

- **https://tinyurl.com/CSCI3753**